数据科学基础
(With Python)

分类

Lazy vs Eager学习

- **Lazy学习: 简单存储训练样本,并不构造一般化的模型,直到需要分类新实例时才分析它与存储样本的关系.**
  - **训练快,预测慢**
  - **例如: kNN分类**
- **Eager学习:给定训练集,构造分类模型,然后才对新实例进行分类.**
  - **训练慢,预测快**
  - **例如:决策树,SVM**

kNN简介

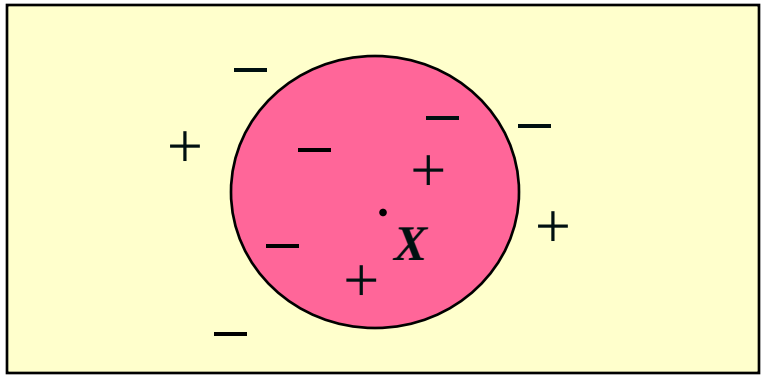
- 一个明显的分类算法:存储所有训练样本,将未知类别样本与内存中样本对比,如果有完全匹配的,则将该已知样本的类别标签赋予这个未知样本
  - 绝对准确,但未知样本可能没有匹配
- 改进:不求绝对相同,只要求最相似.
- 1950s提出,1960s才流行
  - 因为计算密集

kNN分类

- 样本:视为n维空间的点
- 训练集:n维模式空间
- 分类器:搜索与X最近的k个点(近邻)
  - "近":利用欧几里德距离

$$dist(X_1, X_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}$$

  - 未知实例的类别 = k个最近邻中最多见的类别



kNN的讨论(1)

- 也可用于连续值预测: 返回k个最近邻的平均值
- 通常要对特征的值进行标准化
  - 例如Min-Max标准化
- 对类别型数据
  - 简单方法是比较未知样本与训练样本的对应特征值是否相同:相同距离为0,不同距离为1
- k的选择:从k=1开始实验,评估正确率,选择最合适的k

kNN的讨论(2)

- 特征加权
- 可通过删除最无关特征等方法解决
- 距离加权
  - 对k个最近邻与X的距离进行加权
  - 距离越近,权重越大 $w \equiv \frac{1}{d(X, x_i)^2}$
- 求k个最近邻的平均值可免受噪声影响

KNeighborsClassifier(1)

- kNN投票分类器
- 参数:
  - n_neighbors:近邻个数(缺省5)
  - weights:权值函数.缺省值'uniform'表示所有点具有相等权值;'distance'表示权值为距离倒数,越近的近邻影响力越大.
  - metric:距离度量.缺省值'minkowski'
  - p:minkowski距离中的指数(p=1即曼哈顿距离,p=2即欧氏距离)

KNeighborsClassifier(2)

- 方法
  - **predict(X):预测类标签**
  - **predict_proba(X):预测概率**
  - **fit(X, y): 拟合训练集**
  - **score(X, y)在测试集上的平均正确率**

例:kNN分类

```
X = [[0], [1], [2], [3]]
y = [0, 0, 1, 1]
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X, y)
print knn.predict([[1.1]])
[0]
print knn.predict_proba([[0.9]])
[[ 0.66666667  0.33333333]]
```

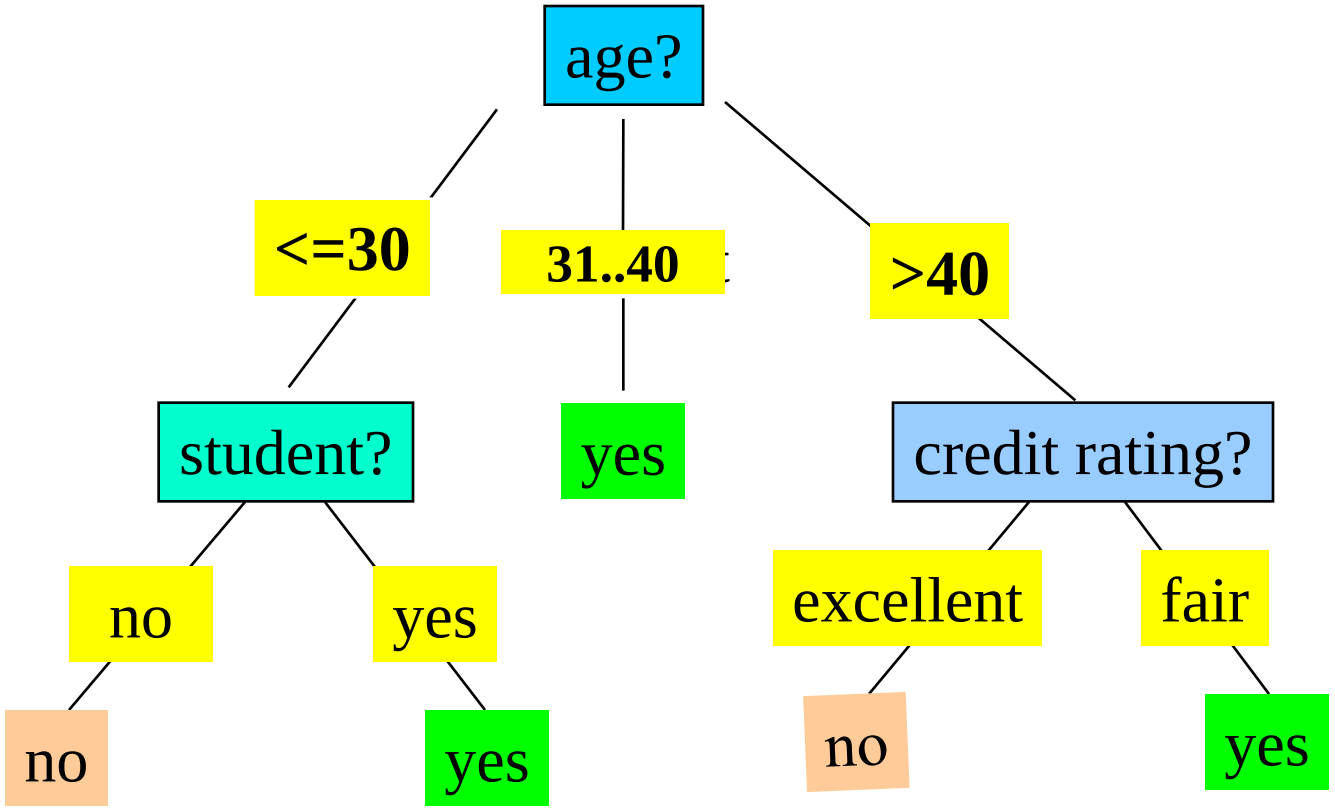
kNN案例

- 附文档

决策树分类

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

决策树


age?
<=30
31..40
>40
student?
yes
credit rating?
no
yes
excellent
fair
no
yes
no
yes

决策树算法

- 贪婪算法:自顶向下递归构造
  - 开始时, 所有训练样本位于根结点
  - 选择划分特征: 能将当前样本归入各自类别的最佳特征
  - 基于所选特征对样本进行划分,构造子结点
  - 对各子结点重复以上步骤,直至满足停止条件

特征选择

- 最佳划分: 能把D划分为若干个"纯的"子集,即集合中样本属于同类别.
  - 选择特征A
  - 选择对特征A的划分
    - 离散值:每个值对应一个分支,下层不再考虑A
    - 连续值:确定一个划分点p(例如从每一对相邻值的中点里选择最佳),然后划分为A≤p和A>p两个分支
      - 或者预先离散化
    - 离散值的二叉划分:所有值划分为两个子集

停止条件

- 满足如下任一条件时划分停止
  - 节点上的所有样本属于同一个类别
  - 没有剩余特征对本结点数据进一步划分.
    - 这时可利用占多数的类作为叶结点标签
  - 结点没有样本
    - 这时可利用**D**中的多数类别作为叶节点标签

数据集纯度:熵

- 若数据集D 包含来自n 个类别的样本, 则信息熵定义为

$$Info(D) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

  - p_i是类别C_i 的概率,可估计为频率$|C_{i,D}| / |D|$
  - D中样本同属一类时熵为0(最纯)
  - D中样本各类都有而且各类样本数相等时最不纯.
  - n=2时(类P和类N各有p和n个样本)的熵

$$I(p,n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

数据集纯度:熵

- 根据特征A将D划分为v个子集之后的熵

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

- 信息增益

$$Gain(A) = Info(D) - Info_A(D)$$

- 选择带来最大Gain(A)(最小$Info_A(D)$)的特征A对D进行划分
- ID3采用此方法

例:利用熵选择特征(1)

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

例:利用熵选择特征(2)

- **yes和no两类各有9和5个样本**

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$$

- **如果选择age来划分:**

$$Info_{age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.694$$

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

- **信息增益:** $Gain(age) = Info(D) - Info_{age}(D) = 0.246$

例:利用熵选择特征(3)

- 同理可对其他特征计算信息增益

$$Gain(income) = 0.029$$

$$Gain(student) = 0.151$$

$$Gain(credit\_rating) = 0.048$$

- 所以在根结点上选择age进行划分

增益率

- 信息增益偏向于具有大量值的属性
  - 如:具有唯一性的特征,划分最纯,信息增益最大,但对预测其他样本没有用处
- C4.5利用增益率来解决此问题:考虑代价

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2(\frac{|D_j|}{|D|})$$

GainRatio (A) = Gain(A)/SplitInfo$_A$(D)

  - 例:|D|=4,用A分为4个子集,用B分为2个子集,且都是纯的,信息增益相同,但前者划分信息为2,后者为1

例:增益率

- 对income计算增益率(划分为low, medium, high)

$$SplitInfo_{income}(D) = -\frac{4}{14}\times\log_2(\frac{4}{14}) - \frac{6}{14}\times\log_2(\frac{6}{14}) - \frac{4}{14}\times\log_2(\frac{4}{14})$$
$$= 0.926$$

GainRatio(income) = 0.029/0.926 = 0.031

- 同理可算出GainRatio(age)=0.156等
- 选择具有最大增益率的特征

数据集纯度:gini指数(1)

- 若数据集**D** 包含来自**n** 个类别的样本, 则**gini**指数定义为

$$gini(D) = 1 - \sum_{i=1}^{n} p_i^2$$

  - p_i是**D**中样本属于类别C_i 的概率,可估计为频率$|C_{i,D}| / |D|$
  - **D**中样本同属一类时最纯:**gini=0**
  - **D**中样本各类都有而且各类样本数相等时最不纯:**gini= 1- 1/n**

数据集纯度:gini指数(2)

- 若数据集**D** 根据**A**划分成子集**D1**和**D2,** 则划分后的纯度定义为两个子集的加权平均纯度

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

- 纯度变化

$$gini(D) - gini_A(D)$$

- 选择具有最小$gini_A(D)$的特征**A**
  - 需要考虑**A**的所有可能的划分点**(**即产生不同的D_1和D_2**)**

例:用gini指数选择特征

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

例:用gini指数选择特征(1)

- D中有9个实例属yes类, 5个实例属no类

$$gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$$

- 考虑用income划分D:划分为{low, medium}(10个实例)和{high}(4个元组)时

$$gini_{income \in \{low, medium\}}(D) = \left(\frac{10}{14}\right) Gini(D_1) + \left(\frac{4}{14}\right) Gini(D_2) = 0.45$$

- 同样计算划分为{low,high}|{medium}和{medium,high}|{low}时的gini指数, 可知income的最佳划分是{medium,high} | {low}, gini=0.30

例:用gini指数选择特征(2)

- 同样地对其他特征计算gini:
  - age的最佳划分:gini=0.375
  - student的最佳划分:gini=0.367
  - credit_rating的最佳划分:gini=0.429
- 因此,选择income作为根节点上的分裂属性.
  - 注意不同于信息增益度量!

特征选择度量的比较

- 实际中都有较好结果, 但也都有偏向性
  - 信息增益(ID3)偏向于多值属性
  - 增益率(C4.5)偏向于不平衡划分,其中一个划分远小于其他划分
  - Gini指数(CART)偏向于多值属性且当类别很多时难于使用
- 还有其他度量,但没有哪个是显著优于其它的

过拟合与剪枝

- 过拟合:许多分支反映了训练数据中的噪声或异常值
- 剪枝:利用统计量删除最不可靠的分支
  - 前剪枝: 尽早停止树的生长,使结点成为叶子
    - 如果划分导致某种度量(信息增益, gini指数等)低于预定的阈值, 就停止划分
    - 难以选择合适的阈值
  - 后修剪: 从完全生长的树中删除分支
    - eg. CART的代价复杂度基于叶节点数目和错误率,比较剪枝前后复杂度的变化,不大则可剪

DecisionTreeClassifier(1)

- 参数:
  - criterion:缺省为"gini",可指定"entropy"
  - splitter:结点划分策略,缺省为"best",可指定"random"(最佳随机划分)
  - max_features:寻求最佳划分时要考虑的特征数目
  - max_depth:树的最大深度.缺省为扩展到所有叶节点都是纯的或者所有叶节点包含的样本数低于min_samples_split参数的规定.
  - min_samples_split:需要划分的最小样本数

DecisionTreeClassifier(2)

- 属性:
  - **feature_importances_:特征重要性**
  - **min_samples_leaf:叶节点上至少应该包含的样本数**
- 方法
  - **predict_proba(X)预测输入样本X的类概率**
  - **fit(X, y)**
  - **predict(X)**
  - **score(X, y)平均准确率**

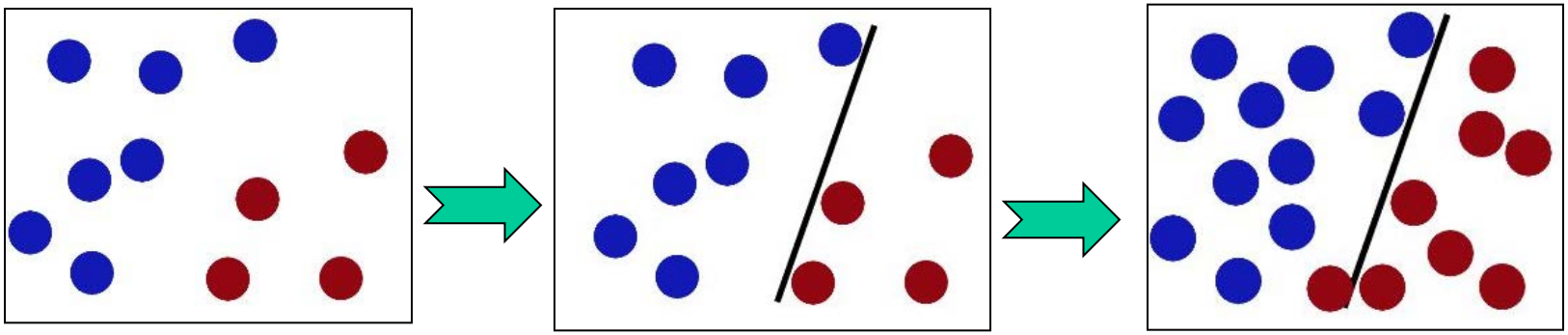
例:决策树分类

```
from sklearn.datasets import load_iris
from sklearn.cross_validation import
  cross_val_score
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()

clf = DecisionTreeClassifier(random_state=0)

cross_val_score(clf,iris.data,iris.target,cv=10)
array([1.        , 0.93333333, 1.        , 0.93333333, 0.93333333,
       0.86666667, 0.93333333, 1.        , 1.        , 1.        ])
```

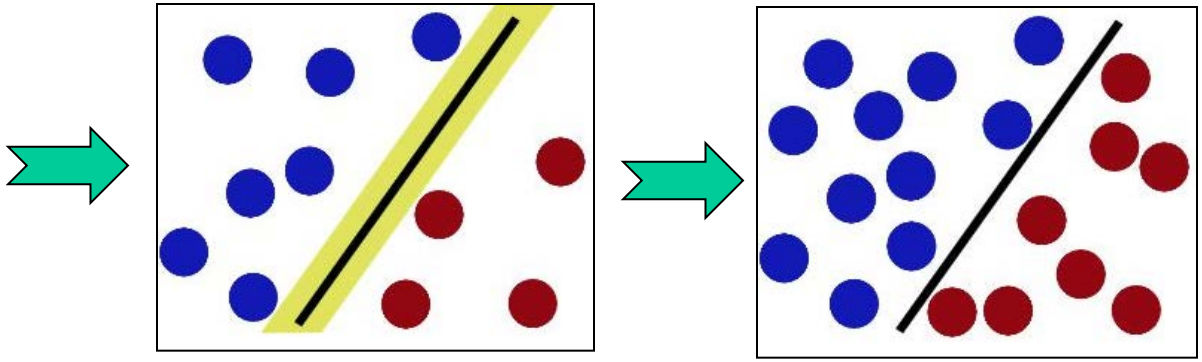
决策树分类案例

- 附文档

SVM直观思想:线性分隔

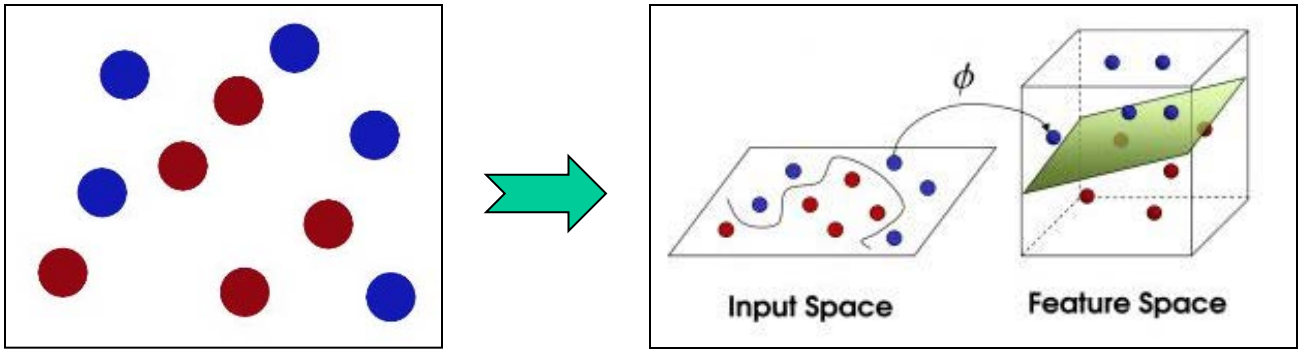


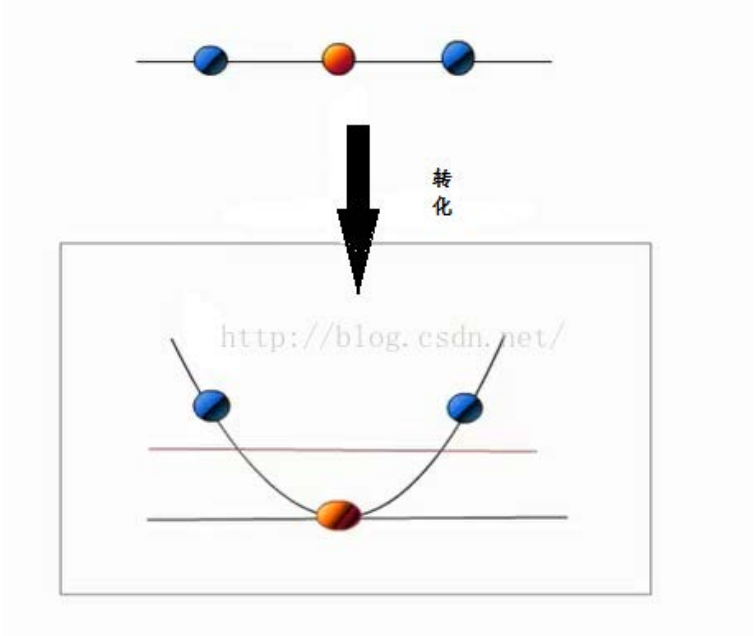
margin小



margin大

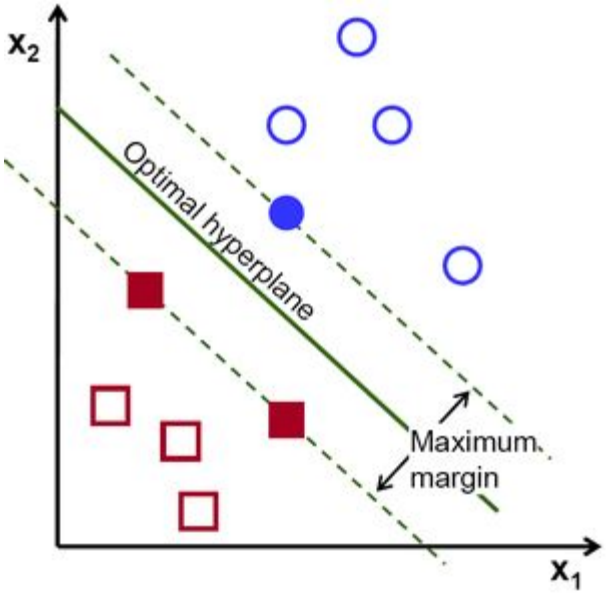
SVM直观思想:非线性分隔





SVM原理(1)

- 搜索分隔超平面将不同类别样本分隔开
- 所有分隔超平面中选择margin最大的(MMH)
- 落在margin两侧边缘上的样本称为支持向量
- 支持向量机根据支持向量的信息对未知类别数据进行分类



SVM原理(2)

- 分隔超平面:$W^T X + b = 0$
  - W是权值$(w_1,...,w_n)$, n是特征数目,b是标量
- 二维$X=(x_1,x_2)$,分隔超平面是直线

  $w_0 + w_1x_1 + w_2x_2 = 0$
  - 落在分隔超平面上方的点(+1类)都满足

    $w_0 + w_1x_1 + w_2x_2 > 0$
  - 落在分隔超平面下方的点(-1类)都满足

    $w_0 + w_1x_1 + w_2x_2 < 0$

SVM分类

- 训练所得MMH可写为决策函数

$$d(X^T) = \sum_{i=1}^{k} y_i \alpha_i X_i X^T + b_0$$

  - y_i是支持向量X_i的类标签(+1和-1)
  - α_i和b_0是优化过程得出的参数
  - k是支持向量的个数。
  - X^T是测试样本
- X^T代入方程,检查结果符号:符号为正则落在MMH上方,预测+1类;符号为负则落在MMH下方,预测为-1类

SVM性能

- 训练时间长
- 高度准确,因为能对复杂非线性判定边界进行建模,较少过拟合
- 可用于线性和非线性数据的分类
- 分类器的复杂度由支持向量的个数刻画而不是数据的维度
  - 如果其他训练样本都删除,重新训练,将得到相同的分隔超平面

非线性分隔

- 两步:
  - 利用非线性映射将原始数据转换到更高维空间
  - 在新空间中搜索线性分隔超平面
    - 最大**margin**超平面对应于原数据空间中的一个非线性分隔超表面.

例:非线性分隔

- 例如:3维数据$X=(x_1,x_2,x_3)$可映射到6维空间数据$(x_1, x_2, x_3, x_1x_1, x_1x_2, x_1x_3)$
- 在新空间中找出判定超平面

$$d(z)=WZ+b$$

解得W和b后然后代回,即得原空间的非线性二阶多项式

$$d(Z) = w_1z_1+w_2z_2+w_3z_3+w_4z_4+w_5z_5+w_6z_6+b$$
$$= w_1x_1+w_2x_2+w_3x_3+w_4(x_1^2)+w_5x_1x_2+w_6x_1x_3+b$$

映射转换的问题

- 两个问题
  - 如何选择非线性映射φ?
  - 计算代价很高:大量点积计算
- 在新高维空间搜索线性SVM时训练样本仅以点积形式$\varphi(X_i)\cdot\varphi(X_j)$出现,存在数学技巧避开这些点积计算!

核函数

- 在转换后的数据上计算点积等价于对原始数据计算一个核函数,即

$$K(X_i,X_j) = \varphi(X_i)\cdot\varphi(X_j)$$

  - 训练中所有的点积计算都可替换成核函数
  - 核函数计算在维数低得多的原空间中进行
  - 从而无需进行实际转换,只要知道映射φ是什么并找出相应的核函数即可

几种核函数

- 线性核函数:$K(X_i,X_j) = (X_i \bullet X_j)$
- 高斯径向基函数(RBF)核

$$K(X_i,X_j) = e^{-\|X_i - X_j\|^2/2\sigma^2}$$

- 多项式核:$K(X_i,X_j) = (X_i \bullet X_j + 1)^d$
- Sigmoid核:$K(X_i,X_j) = \tanh(\kappa X_i \bullet X_j - \delta)$
- 不同的核导致不同的非线性分类器
  - 实际中选择哪一个核一般不会带来准确率的很大不同

二类vs多类

- SVM处理二分类问题
- 多类问题可以采用one-vs-rest方法:
  - 给定m个类,为每个类训练1个分类器,共有m个分类器
  - 分类器k对类k返回正值,对其他类返回负值
  - 对数据的预测是具有最大正距离的类
- 还可采用one-vs-one方法:
  - 为任意两个类构造一个SVM
  - 对未知样本分类时,选择得票最多的类

sklearn.svm.LinearSVC

- 类似参数kernel='linear'的SVC,但LinearSVC对大数据集更有可伸缩性
- 对多类别采用one-vs-rest方式
- 方法:
  - fit(X, y)
  - decision_function(X):样本与超平面的带符号距离
  - predict(X)
  - score(X, y)

sklearn.svm.SVC

- **kernel:缺省为'rbf',可指定'linear', 'poly', 'sigmoid', 'precomputed'等**
- **对多类别采用one-vs-one方式**
- **方法:**
  - **fit(X, y)**
  - **decision_function(X):样本与超平面的带符号距离**
  - **predict(X): 返回1或-1**
  - **score(X, y)**

sklearn.svm.SVC

```
import numpy as np
X = np.array([[-1, -1],
              [-2, -1],
              [1, 1],
              [2, 1]])
y = np.array([1, 1, 2, 2])
clf = SVC()
clf.fit(X, y)
print(clf.predict([[-0.8, -1]]))
[1]
```

SVM案例

- 附文档

End